Follow that Robot

CS39440 Major Project Report

Author: Alex R. Hine (ALH72@aber.ac.uk)

Supervisor: Dr Fred Labrosse (ffl@aber.ac.uk)

April 2021

Version 1.0 (Draft)

This report is submitted as partial fulfilment of a BSc degree in Computer Science (GH76)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, UK

Declaration of originality

I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Registry (AR) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work, I understand and agree to abide by the University's regulations governing these issues.

Name Alex R. Hine

Date: 01/04/2021

Consent to share this work

By including my name below, I hereby agree to this project's report and technical work being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name Alex R. Hine

Date: 01/04/2021

Acknowledgements

I am grateful to all those who have been so supportive and helpful at the university including my supervisor Fred Labrosse and my fellow classmates who always where there to help me if I got confused or misunderstood.

I would like to thank my parents for taking me in during the second Lockdown and letting me stay well past the holidays and my brother who, while more of a hindrance at times, helped give me motivation to finish my work.

Abstract

In robotics a huge part of it is telling a robot where to move, in factory conditions it may be telling the arm of a robot to move to put a part in place or in rovers and other autonomous vehicles it may be moving the rover from point A to B. Now this is not a difficult feat for the most part, you can go about it many ways from manually controlling the robot to complicated mapping and point navigation, but the issue may arise when you have many robots, or a swarm. Programming all to do the same thing may be difficult and time consuming so we need another solution.

So what if, for example, we were talking about rovers on mars, building habitats or some other mission. We need a group of rovers all with cameras to go from point A to point B. it could be quite tedious and time consuming to send the movement commands to all of the robots. My solution would allow the rovers to see the rover ahead and follow it, dynamically detecting and subsequently tracking and following that robot!

If this were to be implemented on these rovers only a single robot would have to be programmed to move to the destination and the rest could just run this routine looking for other rovers and following a distance ahead or behind.

Table of Contents

1. Background, Analysis & Process

This section should discuss your preparation for the project, including background reading, your analysis of the problem and the process or method you have followed to help structure your work. It is likely that you will reuse part of your outline project specification, but as you write this report at the end of the project you should have more to discuss.

1.1. Background

Prior to this project I had worked on a personal robot that shared a few similarities. This robot, Called S.A.M, was designed to track faces with a mounted camera and moving to centre the face in the image. A lot of what I learned in this project I have carried over to my major Project here including: Open CV knowledge, translating image coordinates into movement and other basic functions needed.

I am indeed interested in this project because of its potential in the future, specifically on automation of robots on other planets. But even here on earth there are examples of similar systems, such the platoon model which would make the car follow behind the one in front automatically in a "Assisted self-driving" mode to create a group of vehicles to help increase the capacity of the road. While there are other systems involved with this method (tracking the road and following road markings) the fundamentals are the same, visually identifying a car ahead and following it.

1.2. Analysis

To think about this problem you need break down and identify the key points, for this project they are

1. One robot needs to follow another
2. The robots should be connected, physically or through wireless
3. The following robot needs to be able to see the lead robot
4. The Follower needs to understand what to look for
5. The follower should have some way to understand the position of the lead
6. The follower should be able to move

Given this list of points a few are solved with just the type of robots used, ones that can move and have cameras are obviously essential addressing points 3 & 6 and we will make the program run independently on the following machine with no input from the other addressing point 2

That leaves us with 3 basic areas to cover: Identification, Tracking and Movement

1.2.1. Identification

The Follow robot needs to be able to identify the lead robot by sight only. It must not communicate with the lead in any way and should be a completely independent system. To achieve this a "pattern match" method seems most appropriate, what this means is that an image or pattern will be passed as the template then the program will scan across an image (or frame) and match it to the template then creating a heatmap of where the most likely area is. To do this A library such as Open CV would be ideal.
Open CV [OCV] is an image manipulation library that allows for many different processes to be run on an image including template matching with various methods.

Template matching is a method of finding one image in another. It works by taking the template and scanning it across the image checking each pixel against the other, giving a weight between 0 and 1 to how closely they match. Below are the methods OCV comes with. Three are built in (and their normalized counterparts) each comparing pixels in their own way.

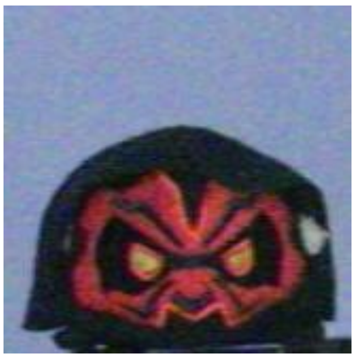


Figure 1 Template Image

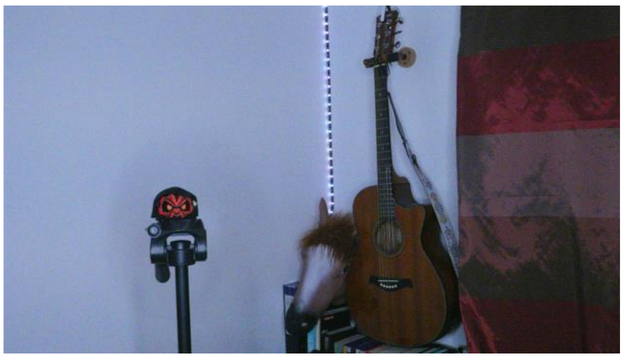


Figure 2 Reference frame

I passed the template and reference image into a temporary program that tested each of the methods and output their images into the following:

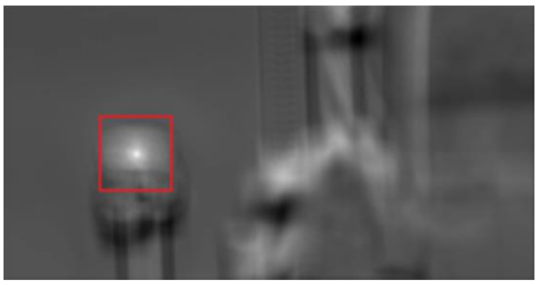


Figure 6 CCOEFF

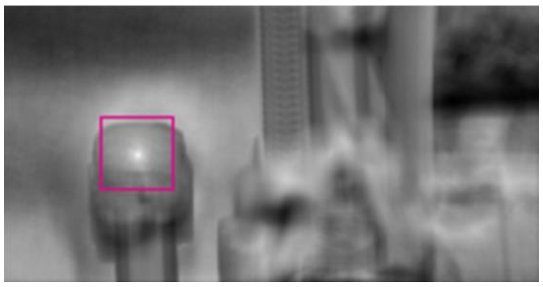


Figure 6 CCOEFF Normalized

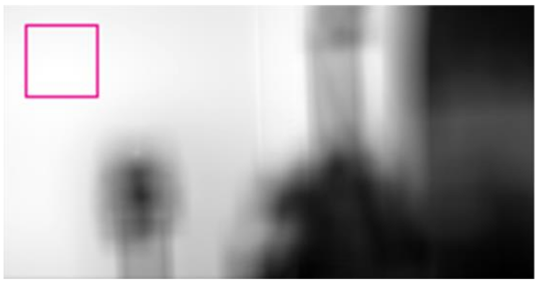


Figure 7 CCORR

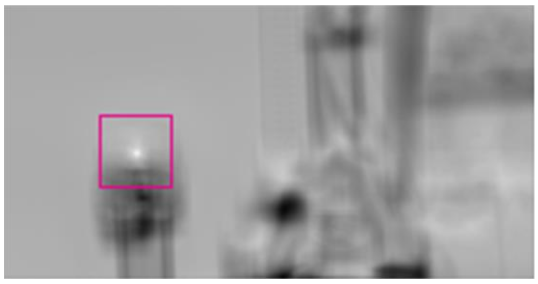


Figure 7 CCORR Normalized

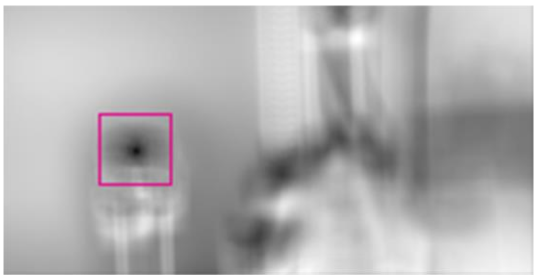


Figure 8 SQDIFF

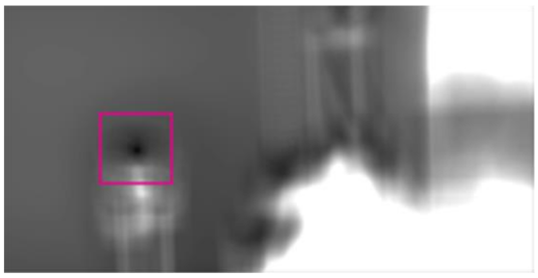


Figure 8 SQDIFF Normalized

As we can see all but one of the methods correctly identified the template. That being said each was no equal, from a time perspective each took a different amount of time.
CCORR was the fastest followed by CCOEFF and SQDIFF which were very similar. with each of their normalized versions taking longer still. While the fractions of a second difference may

seem trivial when going frame by frame the difference is very noticeable. Overall I found SQDIFF to fail more often than CCOEFF in my tests and as such will use CCOEFF.

1.2.2. Tracking

The Follow robot should track the lead robot through vision, knowing the position and scale in frame will allow us to calculate useful information such as distance and which way to turn. OCV can be used here as well as the bounding box created by template matching can be used to gather all that data however, template matching is slow and inefficient, meaning that as the lead vehicle moved (potentially quite quickly) it could be too slow to keep up. To counter this we could use another of OCVs functions called "Object tracking" that will take a bounding box then track the movement of the pixels in that box. This speeds up the processing massively as the program will not have to scan the whole image any more just the immediate surrounding area.

There are eight implementations [1] that are given to OCV and subsequently us when using object tracking, each one comes with benefits and failings, bellow I compiled a table that shows some of the key points

Tracker Name	Speed	Scaling	Accuracy	Notes
Boosting Tracker	Slow	No	Poor	Legacy tracker
MIL	slow/Moderate	Yes	Moderate	
KCF	Fast	No	High	Better at reporting Failure then the previous two
CSRT	slow	Yes	High	
Median Flow	Moderate	Yes	High [under slow movement]	Works well when motion is predictable
TLD	Moderate	Yes	Moderate	Good at tracking though occlusion but can give many false positives
MOSSE	Fast	No	Poor	
GOTURN	??	??	??	Requires external file, dependent on system and deep learning

In my project we need the tracker to be able to scale with the object to show distance so any tracker that does not inherently do this is out. Unfortunately, this means getting rid of KCF which is the highest recommended due to its speed and accuracy, and from my tests I can say that it is indeed the most accurate with least noise, but scaling is a necessity.

This taken into account I feel that median flow would be the ideal method as it give the best balance between speed and accuracy, the note about it only working well under predictable movement is fine as the lead robot should not move to erratically.

1.2.3. Movement

The final part would be the movement, translating the position from the camera into commands that the robot would understand. ROS takes commands for the rotation and acceleration; these should be easily calculated from the size and position of the bounding box gained from the tracker.

1.3. Open CV

Open CV will be essential in this project. It is a library designed to deal with image manipulation such as the matching and tracking described above. In addition the library gives options to visualize and display to a user in a far more friendly way then the terminal by displaying frames and video with text and basic shapes drawn on, such as the bounding box being drawn on the frames and displayed to the user so they can understand what the robot sees and is computing.

In addition to this my familiarity to Open CV from previous personal projects will assist greatly in this project.

1.4. ROS

ROS stands for "Robot Operating System" it is the way most robots, including the ones I will be using operate. The program allows for a set of code to interact with a robot. This is not just a system used for education purposes however, ROS is an industry standard being used by company such as Google and Amazon. ROS is a Unix-based application and as such will only run on Linux based systems (with some ported exceptions however these are not officially supported).
There are 3 main distributions of Linux currently supported: Kinetic[1], Melodic and Noetic, the newest of which, Noetic, will be the one I use as it is the most up to date and has the greatest support.

ROS works on a node-based system. Nodes are a way of computing various parts of a robot, in any system there are likely many nodes, things such as wheels, arms, rangefinders etc can all have individual nodes but also code such as pathing and image processing may have their own nodes as well [2]
Nodes can communicate by passing simple messages such as integers or boolean etc but the common way for Nodes to interact is through Topics. A topic is a Message routing system that allows one node to "publish" a message to the topic, it does not necessarily need a recipient, however if another node were to subscribe to this topic it would receive the information. Commonly wheels are set by the robot to be listening to a specific topic, which code could publish to in order to drive the wheels.
There are other more complicated computation devices but for the scope of this project are most likely overkill and thus unnecessary.

1.5. Programming language

I personally am comfortable using a number of languages for this project: C++, java, or Python however only one can be chosen, and I believe I will use Python for this project.

While open CV works with all three of the above, ROS only communicates with C++ and Python and seeing as my pervious projects that used Open CV where in python, I feel like my experience with the two will aid me in this endeavour.

[1] ROS Kinetic is no longer officially supported as of April 2021 [5]

1.6. Process

The model I used for this project was a combination of the “V-Model” and the “waterfall Model” as for each of the above-described sections I planned out everything then moved to coding then checking go back and forth as necessary once each step was complete, I would move to the next and repeat the planning etc until I had three sections that worked and just needed to be brought together in one final piece.
The reason I decided to do this in this way was to help me focus on each individual point and make it the most adaptable and work robustly. If the tracking is working well then, any issues that arrive during the movement can be attest to that code and not the code from previous sections.
In addition this method gives me three sets of code that can run independently of each other and be reused later on in other projects or be easily adapted for different hardware.

1.7. Version Control and Backups

When it comes to version control and backups I am, ashamedly, quite bad at both. My workflow usually breaks the needs of version control, while I may make old files that get passed on when the code is no longer used or I want to refactor my code for clarity or neatness, my workflow is a mix of small, short bursts mixed with huge changes which I usually do not look back on unless exceptional circumstances. I like to look back on mistakes, especially on code I wrote, and try to understand how and why they went wrong instead of just reverting back to an old version.

When it comes to backups, I have to say again I am lacking, I am not a user of online repositories like GIT etc and so I use physical backups on an internal spare drive and (for this project was important) an external backup drive.

2. Design

2.1. Overall Architecture

The design of this robot needs to accomplish 3 main points: Find the given template, track that image, and follow the lead robot a high-level flow chart would be as follows

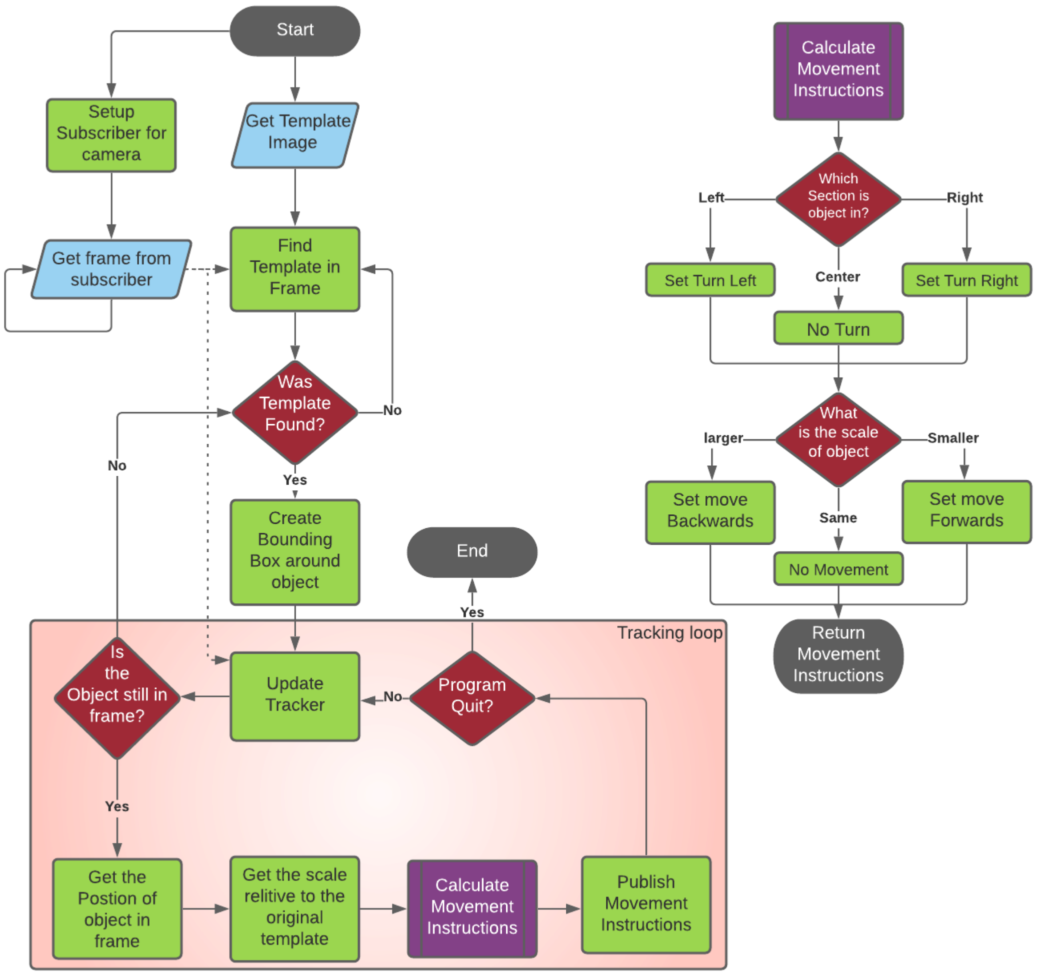

Start
Setup Subscriber for camera
Get Template Image
Get frame from subscriber
Find Template in Frame
Was Template Found?
No
Yes
Create Bounding Box around object
End
Yes
Tracking loop
Is the Object still in frame?
Update Tracker
No
Program Quit?
Yes
Get the Postion of object in frame
Get the scale relitive to the original template
Calculate Movement Instructions
Publish Movement Instructions
Calculate Movement Instructions
Which Section is object in?
Left
Right
Center
Set Turn Left
Set Turn Right
No Turn
What is the scale of object
larger
Smaller
Same
Set move Backwards
Set move Forwards
No Movement
Return Movement Instructions

Figure 9 Flowchart of intended process

This shows the way I hope to make the process work, key to note here that the subscriber section is run independently from the main code, that is because the subscriber is constantly pushing and updating this part independently form the main flow once set up.

This meets all three of the points I laid out before as the template is taken and checked against the frame and when found the program enters the tracking loop, here the bounding box is updated and used to calculate the movement of the robot.

2.2. Detailed Design

Given this flow I intend to follow I decided to make the project a Object Oriented Project, Objects will be useful as there are many places that should be reused and called upon at various places. Below is the class diagram I intend on creating.

2.2.1. Class Diagram

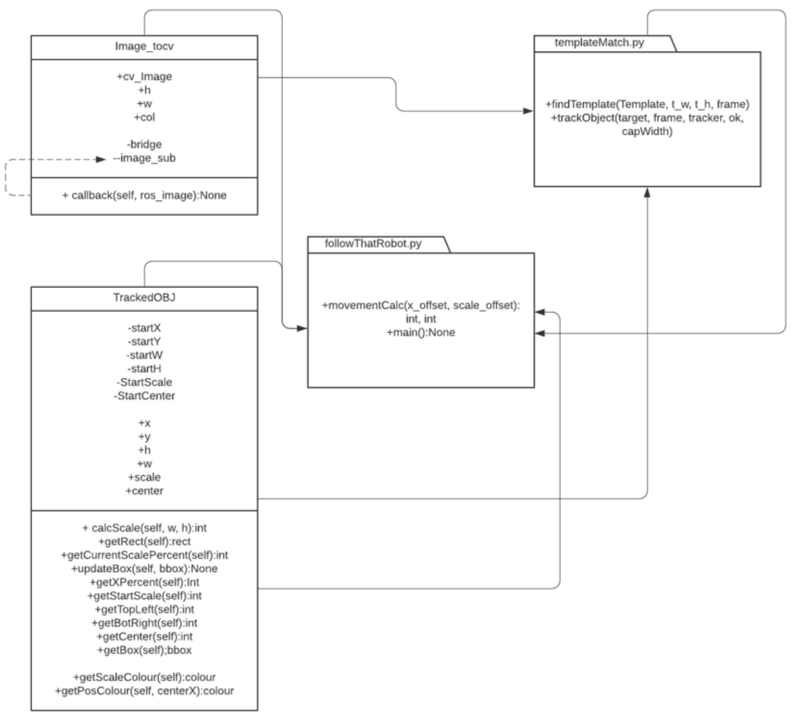

Image_tocv
+cv_Image
+h
+w
+col
-bridge
--image_sub
+ callback(self, ros_image):None
templateMatch.py
+findTemplate(Template, t_w, t_h, frame)
+trackObject(target, frame, tracker, ok, capWidth)
followThatRobot.py
+movementCalc(x_offset, scale_offset): int, int
+main():None
TrackedOBJ
-startX
-startY
-startW
-startH
-StartScale
-StartCenter
+x
+y
+h
+w
+scale
+center
+ calcScale(self, w, h):int
+getRect(self):rect
+getCurrentScalePercent(self):int
+updateBox(self, bbox):None
+getXPercent(self):Int
+getStartScale(self):int
+getTopLeft(self):int
+getBotRight(self):int
+getCenter(self):int
+getBox(self);bbox
+getScaleColour(self):colour
+getPosColour(self, centerX):colour

Figure 10 : Class diagram

At first this may seem overly complicated, but it does have some method to it. The program has two main classes and 4 main files, each separated to allow for flexibility if this was made to be adapted to work with other hardware. The two classes "image_tocv" and "trackedOBJ" have their uses, the first holds the image to be used in the calculations, it has the frame hight, width and colour depth, some of which is used. This class is primarily used however to convert the image from the format ROS gives to one that Open CV can use but I will discuss this late in the implementation.
TrackedOBJ is to hold everything important to the object to be tracked, namely it contains the current position and scale of the object and the starting position and scale, this is important as it will help us calculate movement and scale changes. In addition this class does a lot of the calculation on its position in the frame. There are also methods at the end that calculate a colour value for visualization the position later on. This class is also useful as it can be added too if more than one object was needed to be tracked (although that is beyond the scope of this project)

2.2.2. Template Matching

The template matching is a vital part of the program as it what sets the base for all the bounding boxes, how I wish for this to work is: Program takes template, scans for template in frame, then the program displays a box around what it has found, the user then says "Yes, this is correct" or "No, let me show you what is the box"
Selecting the 'Region of interest' [ROI] should be done automatically with the template matching but a second method for doing so with the user selecting the bounding box would also be useful as a redundancy in case the template matching fails.

2.2.3. Robot Instructions

The second flow chart in figure 9 shows the basic understanding of how the movement is calculated. And as shown in the class diagram we can see that the centre and scale is being tracked, finding the scale difference is easy enough as both the initial size and current size are tracked. The centre is not much more complicated as all you need to do is work out half the width of the frame then compare that to the centre of the tracked object.
From here you convert that to a percentage where less than 100% means the object is to the left of frame / smaller and above 100% equals the opposite. From here we can apply this to a small number such as 1 then take away the same for example say the scale and centre percent was “50%:125%” meaning the object is 50% to the left and is 25% larger so apply that to 1 and we get “0.5:1.25” which is good but both are above 0 which is right for the scale but we want the rotation to move the other direction, (negative numbers and positive numbers are moving in opposite directions) so if we take 1 from both numbers we are left with “-0.5,0.25” which is better as now the rotation will turn in the right direction. In addition this method also has the added benefit of moving quicker or slower depending on how far away from the normal it is, this is great because as the robot moves closer to the correct position it slows down, minimising overshooting and subsequently bounce.

2.2.4. Visualization

Visualization, while not strictly necessary for the project, is a vital part of the design. I believe that the user should be understand what the robot is thinking. To this end I will have the code display a window showing the output of the camera on the robot overlayed with the bounding box and centre dot of what the robot is currently tracking. In addition as I described above, I intend on having the bounding box and centre dot be coloured to show its position in frame. in addition other useful information should be able to be toggles such as the frames per second and debug information such as if the tracking has failed to give the user a visual cue.

3. Implementation

3.1. Image Handling

3.1.1. ROS to OCV

The vision and image handling of the robot is essential for my project. As the robot needs to use sight to identify and track the lead vehicle it is vital it works well. In ROS vision is given its own node and topic to publish to, this is ideal for us as setting up a subscriber would be easy and give us the images.

Unfortunately, ROS returns an image in a format incompatible with Open CV which is not useful. However this is not a problem unique to me and there is a library part of ROS called "CV Bridge" that can be used to bridge the two formats, converting the ROS images into ones compatible with OCV. This problem is dealt with by the file "imageConvert.py" in my files. This file contains the class "Image_tocv" a object type that when initialized sets up a subscriber to a valid given robot vision topic. Line 27 in this file is the converter it reads

```
    self.cv_image = self.bridge.imgmsg_to_cv2(Ros_image, "bgr8")
```

where "Ros_image" is the image taken from the subscriber. "bgr8" is the format that is needed for open CV, this line basically converts the image, in addition there is an error checking so if it fails it will display why instead of crashing the code.

3.1.2. User Interface

While not strictly necessary for the project the user should be able to see what is going on in the robot's "brain", so a visual display is probably useful. Displaying this is easy enough OpenCV can display an image in a window updating it. This image can be drawn on, so I decided to draw the bounding box around where the program sees the tracked object. While this is simple as OCV makes this the cost of one line but there is a small but significant part of it all that I am proud of. When displayed in tracking mode I have added some code at the end of the trackedOBJ class that calculates a colour range dependent on the position and scale of the object, a blue border and dot means the object is centred and as it moved away in either direction the colour will slowly blend into either red or green dependent on direction / scale and the closer the colour is to pure red/green then the further from the origin it is. While not useful in the grand scheme of the project this small feature was very helpful in the testing phases and for demonstration & visualization as the user is able to see clearer.

3.1.3. Frames Per Second

Another useful statistic for the user to know is how many frames the program is able to complete per second, the FPS, a higher framerate would mean the code is efficient and not missing out, but a low framerate (lower then 25) is not ideal as that means the code could be skipping frames and missing data. To show the user the frame rate at any point in "followThatRobot.py" on lines 68 and 87 I gather the current time then again and use the difference to see how much time has passed between those two lines being run, in between which is the main subroutines that do the template matching and tracking. Line 89 is where the text is added to the frame[2]

[2] This is usually commented out to declutter the screen

3.2. Template Matching

3.2.1. Matching the Template to Frame

To begin the template and image must first be brought together and converted. The template matching in open CV requires the input image to be in Gray scale so inside the findTemplate function in "templateMatch.py" firstly the frame is converted into BGR2GRAY which is a type of colour depth for grayscale. Then the template matching occurs looking for the template in img_gray saving the outputted data array to 'res.' The output of the template matching is a matrix that contains a float from 0 to 1 for each pixel on how close it was to the template image. This can then be displayed to the user with an interpreter to give the outputs seen before and below. This is a simple visualization of the matrix where each pixel was assigned a BW value where white is 1 and 0 is black, we can see that it has a spot of brightest points. Seeing as we only want a single match, we do not need to worry about anything but the highest point, we can do this by taking a threshold, which I have set to 1 (so only near perfect matches are selected) and then, using NumPY[3], take the position of these high numbers [line 26] and map them to coordinates that OCV can then use to draw the initial box around for the user to see.

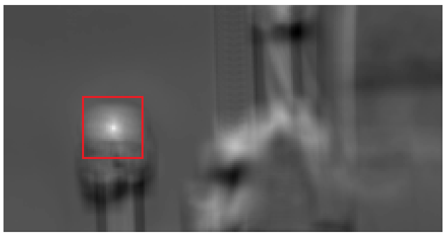


Figure 11 CCOEFF Template matching

The user then sees in the visualization window a box around the found image (if the template is indeed found) and if it is correct then the user presses 'Y' and the process continues however if, for some reason, the match found is incorrect or just not found at all the users can press 'B' which allows the user to then draw a bounding box around whatever object they want tracked and that box will be taken then as a "found template"

3.2.2. TrackedOBJ

Once a template is found it is used to create an object called "target" [lines 24,27]. Target is a trackedOBJ object that is the foundation of the tracking. It contains the information on the x and y coordinate of the top left corner and the hight and width of the box, in addition it calculates the area of the object by multiplying the hight and the width (scale). It also retains the original values of these variables as things such as scale will need to be referenced against the original to show changes in size.
The trackedOBJ also does the calculation for the distance away from the origin that the object may be. It does this two ways, simply for the scale it first takes the area of the original (h*w) then it takes that and divides it by 100 to be multiplied by the area of the updated box.
The full equations for these calculations are [if x,y,w,h are the new values and X,Y,W,H are the original]

$$Scale\ \% = (w * h) * \left(\frac{100}{W * H}\right) \qquad X\ Offset\ \% = \left(x + \left(\frac{w}{2}\right)\right) * \left(\frac{100}{\frac{frameW}{2}}\right)$$

The equations are simple but knowing the percent offset allows for an analogue input to the wheels instead of a binary on / off I can use this to calculate a smoothing equation as described below.

[3] NumPY is a library that deals with matrices and large number arrays

3.3. Tracking

As stated above, it is not resource viable to run template matching for every frame, it would cost too much processing power and be horrible inaccurate as the robot moves and rotates the template may become too dissimilar to not be found. So another method must be used and OCV comes with the ability to do object tracking.
Firstly the tracker needs to be created [ln 59, "followThatRobot.py"] here the type of tracker (median flow) is declared and is created in the variable "Tracker". Then starting after the target has been found the tracker is initialized with the starting cords that the template matching finds. TrackObject [ln 32, "templateMatch.py"] is then called, what this does is update the tracker meaning the program will look at the bounding box and compare it to the previous frame, if the contents have moved it will move the bounding box accordingly then return this. The tracker object is then updated.
One aspect of the tracker is the variable "OK" that is a boolean True or False. This variable is an output that returns true if the tracked object is still found and false if not.
If the tracker is found the box is drawn around where it is found.

3.4. Movement

Movement works by publishing Twist messages to the node created at the start of the program [ln 41, followThatRobot.py"].
To calculate the movement we need to take the calculations from the tracking process and apply them to another equation that can convert the positional data into usable movement commands. The variables we need to publish is the linear and angular velocities to the node. This is calculated in the movementCalc function that takes the offset of the centre and scale of the bounding box.
ROS takes movement variables as an integer value that can be positive or negative, where negative numbers will cause the robot to reverse or turn left and positive numbers the opposite. So to convert a percentage to a usable number first we must divide the number by 100, this will give us a value between 0 and 2 which would be considered always positive so we will need to take 1 from this to move the number down so that the range is now -1 to 1. This would be the end however if the bounding box is 125% scale meaning the lead robot is too close and the follower needs to reverse, if we plug this into the equation, we get 0.25, a positive integer that will mean the robot moves forward and further increase the scale. To fix this we need to multiply the whole equation by -1 this will invert the number to the other side of 0 moving the robot in the right direction, the same is with the angular velocity needing it to be inverted. The resulting equation is then (where x equals the scale or centre offset)

$$Velocity = \left(\left(\frac{x}{100}\right) - 1\right) * -1$$

Now the robot does move in the right directions however due to the nature of the tracking not being 100 percent perfect, there is minor noise from the trackers movement meaning the offset will never truly sit at the origin and always bounce around, and with this system unfortunately this causes the program to overshoot and oscillate around the origin creating larger and larger waves. To solve this I added a simple check after the equation calculation for both velocities that checks if the number returned is in the range -0.1 to 0.1, if so It sets the velocity to 0 stopping the robot and negating the overshot and masking the noise.
The final point is one last check, if the offset passed to the movement calculations is 0 meaning the tracker has lost the object the program sets the velocity to 0 as passing 0 into the calculation returns 1 always (max movement speed) so if the tracking were lost the follower would just begin driving top speed away.

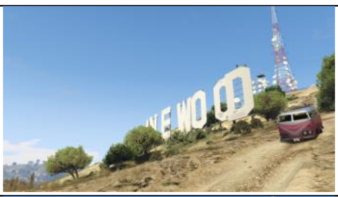
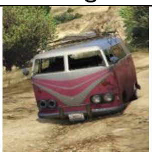
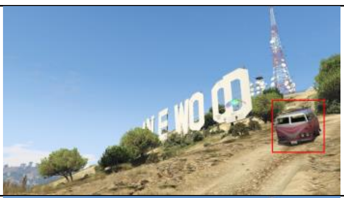
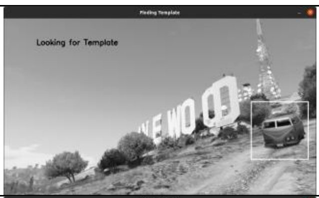
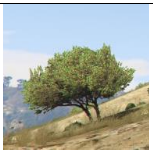
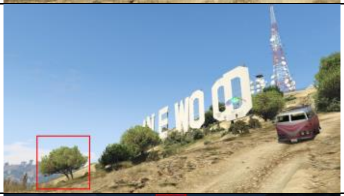
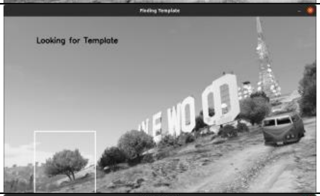
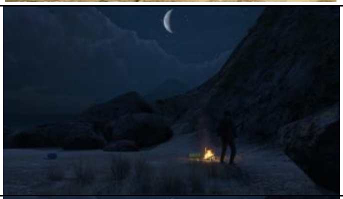
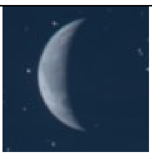
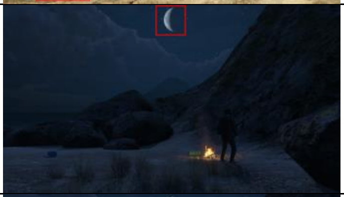
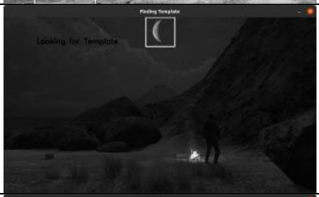
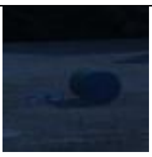
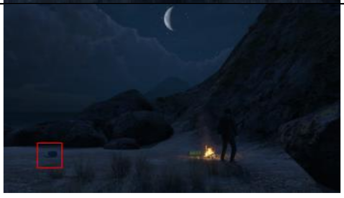
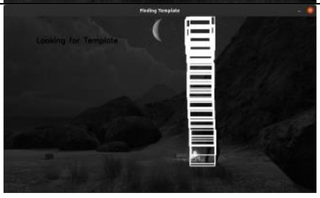
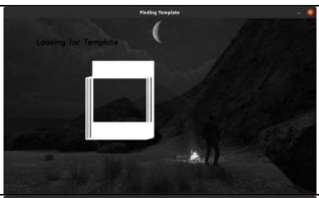
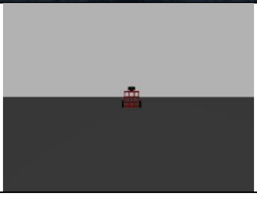
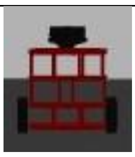
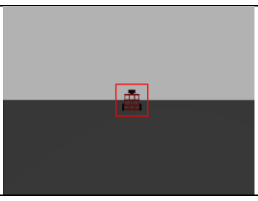
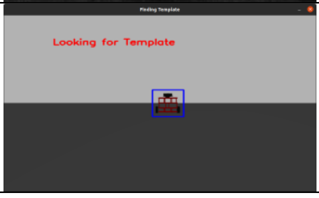
4. Testing

4.1. Overall Approach to Testing

Testing my robot will require testing each of the three systems, the template matching, the tracking, and the movement, luckily each of these systems are independent of each other and can be tested without relying on the previous, with exception to movement which will need the tracking to understand.

4.2. Testing the Template Matching

To test the template matching I need to test if an object is found so I will have to change two things: 1) the input image, to test on a clean and cluttered background for different images, and 2) the template image, how far can I push the template matching.
Bellow is a table with my tests and resulting images (as stated all code is the same to runtime code with the exception of image input path being changed to a static image[4])

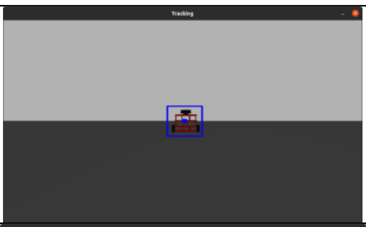
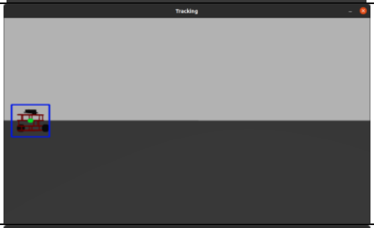
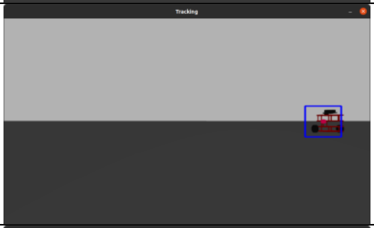
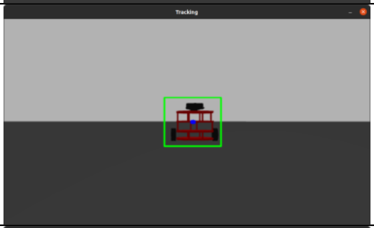
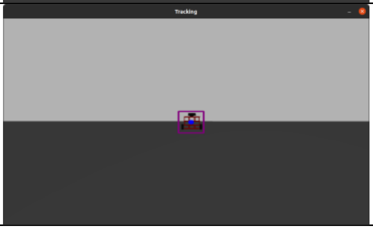
Base Image	Template Image	Expected	Output	Result
			 Looking for Template	Template found
			 Looking for Template	Template Found
			 Looking for Template	Template found
			 Looking for Template	Failed. I believe because the image was too small and vague and too similar to the background to be correctly identified
		Template Not Found.	 Looking for Template	Failed, I believe because while the template was not in the image it still found something, and so the highest chance was taken
			 Looking for Template	Template Found

The tests show that the template matching is not perfect, when given templates that have low contrast the matcher fails, and that if the template is not found then the matcher will match what it can and give false positives. The take a ay is that robot should be in a well-lit area and not to accept the matcher until the true template is found.

[4] All images are my own unless stated otherwise

4.3. Testing the Object Tracking

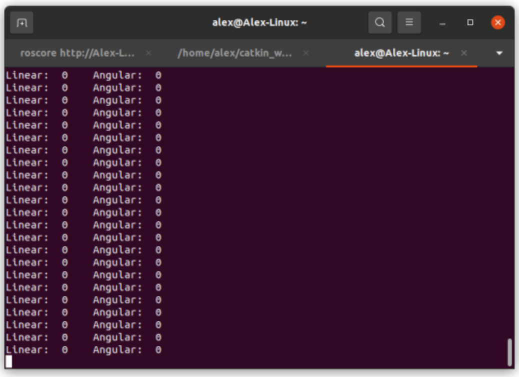
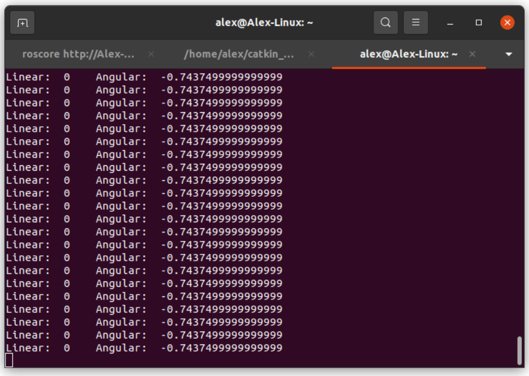
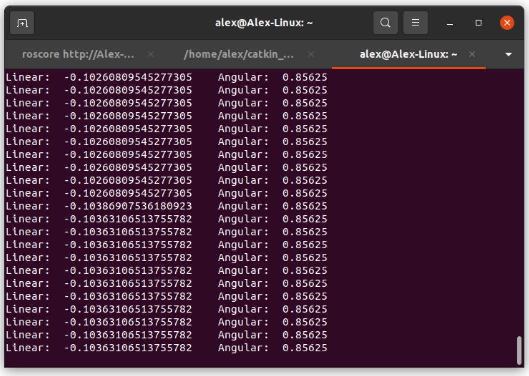
To test the tracking I will be using the colour border system I implemented, while turning off the movement so the robot does not try to correct the error and we can see the program acknowledge the tracked objects changes.

Direction to move	Expected outcome	Output	Result
Object is Centred	Blue box and Dot		Works as intended
Move object to Left	Dot goes Green		Works as intended
Move object to Right	Dot goes Red		Works as intended
Move object Close	Box goes Green		Works as intended
Move object Away	Box goes Red		Works as intended
Fast move Left to Right	Object is tracked	Tracking is lost	The Tracker can't follow an object if it moves across the screen too quickly, at a speed where it would have fully left the bounding box between frames.
Fast move Near to Far	Object is tracked	Tracking continues but change in scale is lost	While the tracking is not lost if the object moves exceptionally quickly forward or backwards the scale does not change

The testing proves that with small, controlled movements between frames like that of a vehicle the tracker can keep up and know where an object is, but with too fast of motion or teleporting an object (in the simulation) the tracker will fail.

4.4. Testing the Movement

To test the movement we will take the tracked objects position and move it. Obviously, I cannot show motion here. Some testing code that displays in the terminal what the linear and angular velocities are this will allow us to see how the program reacts to the movement.

Direction to move	Expected outcome	Output	Result
Object still	No Velocity		Works as intended
Move object to Left	Negative Angular Velocity		Works as intended
Move object to Right	Positive Angular Velocity		Works as intended

alex@Alex-Linux: ~

roscore http://Alex-L... /home/alex/catkin_w... alex@Alex-Linux: ~

```
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
```

alex@Alex-Linux: ~

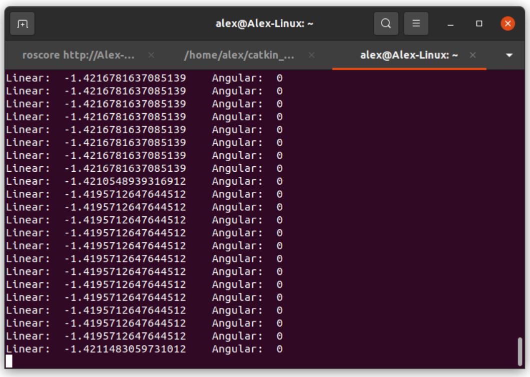
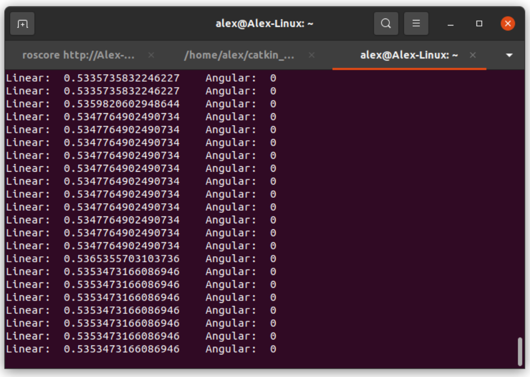
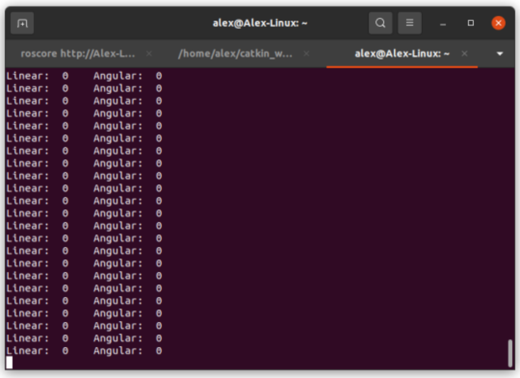
roscore http://Alex-... /home/alex/catkin_... alex@Alex-Linux: ~

```
Linear:  0    Angular:  -0.7437499999999999
Linear:  0    Angular:  -0.7437499999999999
Linear:  0    Angular:  -0.7437499999999999
Linear:  0    Angular:  -0.7437499999999999
Linear:  0    Angular:  -0.7437499999999999
Linear:  0    Angular:  -0.7437499999999999
Linear:  0    Angular:  -0.7437499999999999
Linear:  0    Angular:  -0.7437499999999999
Linear:  0    Angular:  -0.7437499999999999
Linear:  0    Angular:  -0.7437499999999999
Linear:  0    Angular:  -0.7437499999999999
Linear:  0    Angular:  -0.7437499999999999
Linear:  0    Angular:  -0.7437499999999999
Linear:  0    Angular:  -0.7437499999999999
Linear:  0    Angular:  -0.7437499999999999
Linear:  0    Angular:  -0.7437499999999999
Linear:  0    Angular:  -0.7437499999999999
Linear:  0    Angular:  -0.7437499999999999
Linear:  0    Angular:  -0.7437499999999999
Linear:  0    Angular:  -0.7437499999999999
Linear:  0    Angular:  -0.7437499999999999
Linear:  0    Angular:  -0.7437499999999999
```

alex@Alex-Linux: ~

roscore http://Alex-... /home/alex/catkin_... alex@Alex-Linux: ~

```
Linear:  -0.10260809545277305    Angular:  0.85625
Linear:  -0.10260809545277305    Angular:  0.85625
Linear:  -0.10260809545277305    Angular:  0.85625
Linear:  -0.10260809545277305    Angular:  0.85625
Linear:  -0.10260809545277305    Angular:  0.85625
Linear:  -0.10260809545277305    Angular:  0.85625
Linear:  -0.10260809545277305    Angular:  0.85625
Linear:  -0.10260809545277305    Angular:  0.85625
Linear:  -0.10260809545277305    Angular:  0.85625
Linear:  -0.10260809545277305    Angular:  0.85625
Linear:  -0.10386907536180923    Angular:  0.85625
Linear:  -0.10363106513755782    Angular:  0.85625
Linear:  -0.10363106513755782    Angular:  0.85625
Linear:  -0.10363106513755782    Angular:  0.85625
Linear:  -0.10363106513755782    Angular:  0.85625
Linear:  -0.10363106513755782    Angular:  0.85625
Linear:  -0.10363106513755782    Angular:  0.85625
Linear:  -0.10363106513755782    Angular:  0.85625
Linear:  -0.10363106513755782    Angular:  0.85625
Linear:  -0.10363106513755782    Angular:  0.85625
Linear:  -0.10363106513755782    Angular:  0.85625
Linear:  -0.10363106513755782    Angular:  0.85625
```

Move object Close	Negative Linear Velocity		Works as intended
Move object Away	Positive Linear Velocity		Works as intended
Tracking is lost	Velocity should default to 0		Works as intended

```
Linear:  -1.4216781637085139    Angular:  0
Linear:  -1.4216781637085139    Angular:  0
Linear:  -1.4216781637085139    Angular:  0
Linear:  -1.4216781637085139    Angular:  0
Linear:  -1.4216781637085139    Angular:  0
Linear:  -1.4216781637085139    Angular:  0
Linear:  -1.4216781637085139    Angular:  0
Linear:  -1.4216781637085139    Angular:  0
Linear:  -1.4210548939316912    Angular:  0
Linear:  -1.4195712647644512    Angular:  0
Linear:  -1.4195712647644512    Angular:  0
Linear:  -1.4195712647644512    Angular:  0
Linear:  -1.4195712647644512    Angular:  0
Linear:  -1.4195712647644512    Angular:  0
Linear:  -1.4195712647644512    Angular:  0
Linear:  -1.4195712647644512    Angular:  0
Linear:  -1.4195712647644512    Angular:  0
Linear:  -1.4195712647644512    Angular:  0
Linear:  -1.4195712647644512    Angular:  0
Linear:  -1.4195712647644512    Angular:  0
Linear:  -1.4195712647644512    Angular:  0
Linear:  -1.4211483059731012    Angular:  0
```

```
Linear:  0.5335735832246227    Angular:  0
Linear:  0.5335735832246227    Angular:  0
Linear:  0.5359820602948644    Angular:  0
Linear:  0.5347764902490734    Angular:  0
Linear:  0.5347764902490734    Angular:  0
Linear:  0.5347764902490734    Angular:  0
Linear:  0.5347764902490734    Angular:  0
Linear:  0.5347764902490734    Angular:  0
Linear:  0.5347764902490734    Angular:  0
Linear:  0.5347764902490734    Angular:  0
Linear:  0.5347764902490734    Angular:  0
Linear:  0.5347764902490734    Angular:  0
Linear:  0.5347764902490734    Angular:  0
Linear:  0.5347764902490734    Angular:  0
Linear:  0.5365355703103736    Angular:  0
Linear:  0.5353473166086946    Angular:  0
Linear:  0.5353473166086946    Angular:  0
Linear:  0.5353473166086946    Angular:  0
Linear:  0.5353473166086946    Angular:  0
Linear:  0.5353473166086946    Angular:  0
Linear:  0.5353473166086946    Angular:  0
Linear:  0.5353473166086946    Angular:  0
```

```
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
Linear:  0    Angular:  0
```

As we can see the robot responds to all the directions correctly, this is then interpreted by ROS and the robot does move correctly

5. Critical Evaluation

In terms of my project it was a fun one to attempt, like I stated before my previous project shared many similarities, and I certainly learned a lot and improved on my previous design and when I go on to remake my previous project, I surely will take some of my knowledge from this and improve on it again.

Overall I think I hit the main points of the project really well, the exact specification was “A robot that follows another through sight alone” and I have definitely achieved that. The robot does not communicate with any other only gathering information visually and does a good job at keeping the lead robot at a constant distance and within frame. The program works quickly with only rare and minor flaws and is written in a way that allows it to be easily adaptable.

However there where a few features I failed to foresee and therefore would change. First and foremost I am not a fan of simulation, especially gazebo (the simulation program used) I find it difficult to work with and it caused me much unwarranted stress, so if I where to do this again I would definitely work in the real world, this also would have the added benefit of making the code have to be more robust. The double edged sword of the simulation route is the perfection, on one hand the situations are always perfect, there are no external factors like the sun at the time of day, wind or other environmental issues meaning the program will always run the same given the same inputs however the real world is not the same and I feel like maybe my project is worse off having never been tested in these conditions, the need to make the code more robust to deal with them would have improved my code. So one point I would like to change is to do it in real life. Obviously given the year and the pandemic response it was impossible to do this time.

When it comes to features, I wish I could have added, my initial design had a system in place to make the user not have to give any input making the program fully independent. The program would, after having found the same template for a certain time, automatically realise it has found the correct template however due to time restraints and issues with the template matcher not being perfect, I had to cut it. Given the chance to do this again I would have prioritized this feature and not had to scrap it.
Another thing I would change is the issue of the bounding box selector, I learned how to do it in my initial tests but removed the feature to make the program more “independent” then later down the line when that was scrapped, I was asked if adding it back in would help with the matchers if the template was not found but, while it was added in it does not work properly and thus can be useless at times. Given the chance to work on it again I would plan to include this feature from the start and work it into the code so it would work with the program instead of onto of the code.

When it comes to my own shortcomings as I mentioned above my version control is poor and I should really work on that myself. It is not good coding practice

6. Bibliography

[1] A. Rosebrock, "OpenCV Object Tracking," pyimagesearch, 30 July 2018. [Online]. Available: https://www.pyimagesearch.com/2018/07/30/opencv-object-tracking/.

[2] ROS, "ROS Concepts," ROS, 21 06 2014. [Online]. Available: http://wiki.ros.org/ROS/Concepts.

[3] "Open Cv Website," OpenCV, [Online]. Available: https://opencv.org/.

[4] "Ros," Open Robotics, [Online]. Available: Ros.org.

[5] ROS, "ROS Installations," ROS, [Online]. Available: http://wiki.ros.org/ROS/Installation.

7. Appendices

A. Third-Party Code and Libraries

Open CV
ROS Noetic
    CV Bridge

Open CV Library – This was used to deal with all the vision and template matching and object tracking. The version used was v4.5.X. It is a open source project available from OpenCV.org [3]. It is released under the BSD 3-Clause License and is used without modification.

ROS – Ros is the library used to communicate with the robot in any way. The version used was ROS Noetic. The library is open source library released under a BSD licence. It is available from the Open Robotics website Ros.org [4]

B. Ethics Submission

Image not found or type unknown

12/03/2021

For your information, please find below a copy of your recently completed online ethics assessment

Next steps

Please refer to the email accompanying this attachment for details on the correct ethical approval route for this project. You should also review the content below for any ethical issues which have been flagged for your attention

Staff research - if you have completed this assessment for a grant application, you are not required to obtain approval until you have received confirmation that the grant has been awarded.

Please remember that collection must not commence until approval has been confirmed.

In case of any further queries, please visit www.aber.ac.uk/ethics or contact ethics@aber.ac.uk quoting reference number **19277**.

Assesment Details

AU Status
Undergraduate or PG Taught

Your aber.ac.uk email address
alh72@aber.ac.uk

Full Name
Alexander Hine

Please enter the name of the person responsible for reviewing your assessment.
Neil Taylor

Please enter the aber.ac.uk email address of the person responsible for reviewing your assessment
nst@aber.ac.uk

Supervisor or Institute Director of Research Department

cs

Module code (Only enter if you have been asked to do so)
CS39440

Proposed Study Title
Follow that robot

Proposed Start Date
25th January 2021

Proposed Completion Date
01 June 20201

Are you conducting a quantitative or qualitative research project?
Mixed Methods

Does your research require external ethical approval under the Health Research Authority?
No

Does your research involve animals?
No

Does your research involve human participants?
No

Are you completing this form for your own research?
Yes

Does your research involve human participants?
No

Institute
IMPACS

Please provide a brief summary of your project (150 word max)
A robot programmed to visually identify another robot ahead and follow this robot

Where appropriate, do you have consent for the publication, reproduction or use of any unpublished material?
Yes

Will appropriate measures be put in place for the secure and confidential storage of data?
Yes

Does the research pose more than minimal and predictable risk to the researcher?
No

Will you be travelling, as a foreign national, in to any areas that the UK Foreign and Commonwealth Office advise against travel to?
No

Please include any further relevant information for this section here:

Is your research study related to COVID-19?
No

If you are to be working alone with vulnerable people or children, you may need a DBS (CRB) check. Tick to confirm that you will ensure you comply with this

requirement should you identify that you require one.
Yes

Declaration: Please tick to confirm that you have completed this form to the best of your knowledge and that you will inform your department should the proposal significantly change.
Yes

Please include any further relevant information for this section here: